# Fișa 07 C# – utilizare parametri funcții (metode)

1. **Observație**: în limbajul C# funcțiile (metodele) nu se pot defini în afara unei clase.
2. Parametrii unei metode pot fi precedați în C# de câteva cuvinte cheie, numite **modificatori**. Aceștia sunt: **ref**, **out**, și **params** și controlează modul în care parametrii efectivi sunt transmiși unei metode: prin valoare, prin referință și, respectiv, sub formă de listă.
3. **Transmiterea prin valoare** se produce când modificatorii lipsesc și este asemănătoare cu cea din C++. Exemplu:

class test

{

public void schimbVal( int a, int b )

{

int aux = a;

a = b;

b = aux;

}

public void schimbVect( int[] a )

{

//marim toate elementele cu 2

for( int i=0; i<a.Length; i++ )

a[i] += 2;

}

}

class Program

{

static void Main()

{

//definire variabile

test t = new test(); //construim un obiect de tip "test" pentru a putea apela funcțiile din el

int c = 7, d = 3;

int[] v = {1, 2, 3, 4, 5, 6,7 ,8 ,9, 10};

//apel funcții și afișare rezultate

t.schimbVal( c, d );

Console.Write("\nVariabilele c si d dupa apelul functiei schimbVal : " + c + " " + d + "\n");

t.schimbVect( v );

Console.Write( "\nValorile din vectorul v dupa apelul funtiei schimbVect : " );

for (int i = 0; i < v.Length; i++)

Console.Write( v[i] + " " );

//gata

Console.Write( "\n\n\n" );

}

}

Se observă că valorile variabilelor ***c*** și ***d*** din metoda **Main** (din clasa **Program**) nu se schimbă deoarece transmiterea lor către metoda **schimbVal** (din clasa **test**) se face prin valoare. În cazul vectorilor, ca și în C++, transmiterea se face automat prin referință și în consecință toate schimbările efectuate în metoda **schimbVect** asupra vectorului ***a*** se fac de fapt asupra vectorului ***v***, transmis ca parametru. Deci toate valorile din vectorul ***v*** vor crește cu 2 în urma apelului metodei.

1. **Transmiterea prin referință** se face folosind modificatorul **ref** atât în definiția metodei cât și în momentul apelului. Pentru un astfel de parametru, în momentul apelului trebuie să se folosească neapărat o variabilă (deci nu o valoare sau o expresie) ***și*** aceasta trebuie să fie inițializată. Exemplu:

class test

{

public void schimbValRef( ref int a, ref int b ) //parametrii a și b se transmit prin referință

{

int aux = a;

a = b;

b = aux;

}

}

class Program

{

static void Main()

{

//definire variabile

test t = new test(); //construim un obiect de tip "test" pentru a putea apela funcțiile din el

int c = 7, d = 3;

//apel metodă și afișare rezultate

t.schimbValRef( ref c, ref d );

Console.WriteLine("\nVariabilele c si d dupa apelul functiei schimbValRef : " + c + " " + d);

//gata

Console.Write("\n\n\n");

}

}

Se observă că, în acest caz, variabilele ***c*** și ***d*** din **Main** sunt transmise prin referință către metoda **schimbValRef** și orice modificare produsă asupra parametrilor ***a*** și ***b*** din aceasta se va face de fapt asupra lor, deci valorile li se inversează.

1. **Parametrii de ieșire** se definesc cu ajutorul modificatorului **out** și se folosesc pentru a transmite date din interiorul funcției către codul care a apelat-o. Folosirea acestui modificator presupune respectarea următoarelor cerințe:
   1. Modificatorul trebuie să apară atât în definiția metodei cât și în momentul apelului;
   2. Argumentul transmis funcției trebuie să fie obligatoriu o variabilă, dar nu este necesar ca aceasta să fie inițializată;
   3. În interiorul metodei parametrii de ieșire trebuie să fie inițializați înainte de folosire;
   4. Tuturor parametrilor de ieșire trebuie să li se atribuie valori înainte de revenirea din apel.

În exemplul de mai jos se definește o metodă numită *suma\_produs* care primește două valori și furnizează prin intermediul parametrilor de ieșire ***s*** și ***p*** suma și produsul acestora:

class test

{

public void suma\_produs(int a, int b, out int s, out int p) //parametrii s și p sunt de ieșire

{

s = a + b;

p = a \* b;

}

}

class Program

{

static void Main()

{

//definire variabile

test t = new test(); //construim un obiect de tip "test" pentru a putea apela metodele din el

int c = 7, d = 3, suma, produs;

//apel metodă și afișare rezultate

t.suma\_produs( c, d, out suma, out produs );

Console.WriteLine("\nSuma : " + suma);

Console.WriteLine("\nProdus : " + produs);

//gata

Console.Write("\n\n\n");

}

}

1. **Parametrii tablou** se definesc folosind modificatorul **params**. Acesta permite transmiterea către o funcție a zero sau mai mulți parametri efectivi prin intermediul unui singur parametru formal (de tip tablou). În cazul folosirii acestui modificator:
   1. Poate exista un singur parametru tablou în lista de parametri a funcției și acesta va fi ultimul din listă;
   2. Modificatorul **params** trebuie să apară în definiția funcției, dar nu apare în momentul apelului.

În exemplul de mai jos metoda *f* primește o listă de parametri (oricâți) de tip string și îi afișează pe ecran:

class test

{

public void f(params string[] t) //t este un vector cu 0 sau mai mulți parametri de tip string

{

Console.Write("\nparametrii sunt : ");

foreach (string s in t)

Console.Write(s + " ");

}

}

class Program

{

static void Main()

{

//definire variabile

test t = new test(); //construim un obiect de tip "test" pentru a putea apela metodele din el

//apel metodă

t.f( "mama", "tata", "si gigel" );

t.f();

t.f("1", "2", "3", "4", "5");

//gata

Console.Write("\n\n\n");

}

}

1. Scrieți o aplicație de tip consolă care să conțină o metodă cu parametrul ***n*** de tip int. Metoda determină suma cifrelor numărului ***n*** transmis ca parametru.
2. Scrieți o aplicație de tip consolă care să conțină o metodă cu parametrul ***n*** de tip int. Metoda verifică dacă numărul ***n*** transmis ca parametru este prim.
3. Scrieți o metodă care determină cel mai mic divizor prim al unui număr natural ***n*** transmis ca parametru. Folosiți metoda într-o aplicație de tip consolă care să determine numerele prime dintr-un tablou unidimensional cu ***n*** elemente.
4. Scrieți un program care definește o metodă numită *inter* care inter-schimbă între ele valorile a două variabile primite ca parametri. Programul va folosi metoda pentru a sorta crescător valorile dintr-un vector citit de la tastatură.
5. Scrieți o aplicație de tip consolă care să conțină o metodă cu un parametru de ieșire ***n*** de tip int, care va determina inversul lui ***n***. Aplicația va afișa pe rânduri separate valoarea unei variabile transmise funcției înainte de apel și după apel.
6. Scrieți o aplicație de tip consolă care să conțină o metoda cu lista de parametri de tip tablou unidimensional ***x***. Metoda va determina suma elementelor tabloului.
7. Scrieți un program care folosește o clasă numită „Complex” pentru a lucra cu numere complexe de forma a+b\*i. Clasa va implementa metode pentru calculul sumei, diferenței, produsului și a rezultatului împărțirii a două numere complexe. Programul va conține câteva apeluri ale acestor metode pentru a demonstra rezultatele obținute și modul de folosire.
8. Scrieți un program care folosește o clasă numită „Rational” pentru a lucra cu numere raționale de forma a/b (a și b fiind numere întregi). Clasa va implementa metode pentru simplificarea unui număr rațional, calculul sumei, diferenței, produsului și a rezultatului împărțirii a două numere raționale. Programul va conține câteva apeluri ale acestor metode pentru a demonstra rezultatele obținute și modul de folosire.
9. Scrieți un program care definește o clasă numită „test” și o metodă a acesteia numită *calcule*. Metoda va primi ca parametri 0 sau mai multe numere naturale și va afișa suma, produsul și media aritmetică a acestora. Dacă se apelează metoda fără nici un parametru se va afișa pe ecran doar mesajul „Numerele lipsesc!”.
10. *(punct în plus)* Scrieți un program care definește o clasă numită „varianta”. Clasa are un câmp de tip vector cu 6 elemente, numere naturale între 1 și 49, ordonate crescător. Definiți și metodele *nou()* care inițializează cele 6 elemente cu valori noi generate aleatoriu, *ToString()* care transformă vectorul într-un string ce conține cele 6 numere, *Parse(string s)* care primește ca parametru un șir ce conține 6 numere și le introduce în obiectul de tip „varianta” căruia îi aparține. Folosiți astfel definită pentru a genera și afișa pe ecran 3 variante de jucat la loto.
11. *(nota 10)* Rezolvați exercițiul 13 folosind **supraîncărcarea operatorilor** pentru calculul operațiilor cerute și definiți o metodă *ToString()* care să transforme un număr complex într-un string (pentru afișare) și o metodă *Parse(string s)* care să transforme un șir de forma „3+2i” într-un obiect de tip Complex (pentru citire). *(GIYF pentru a vedea ce înseamnă* ***supraîncărcarea operatorilor****). Se acordă doar punct în plus dacă se rezolvă cu supraîncărcare operatori, fără a defini metodele ToString() și Parse().*
12. *(nota 10)* Rezolvați exercițiul 14 folosind supraîncărcarea operatorilor pentru calculul operațiilor cerute și definiți o metodă *ToString()* care să transforme un număr rațional într-un string (pentru afișare) și o metodă *Parse(string s)* care să transforme un șir de forma „3/2” într-un obiect de tip Rațional (pentru citire). *Se acordă doar punct în plus dacă se rezolvă cu supraîncărcare operatori, fără a defini metodele ToString() și Parse().*